

CS161 Final Exam

Do not turn this page until you are instructed to do so!

Instructions: Solve all questions to the best of your abilities. You have **3 hours** to complete this exam. You may use two two-sided sheets of notes that you have prepared yourself. You may not use any other notes, books, or online resources. There is one blank page at the end that you may tear off as scratch paper, and one blank page for extra work. **Please write your name at the top of all pages.**

Tips: Pay attention to the instructions for each problem, and be sure to look at all pages. If you are stuck on a problem, it might be wise to move on to something else and come back to it later.

The following is a statement of the Stanford University Honor Code:

1. *The Honor Code is an undertaking of the students, individually and collectively:*
 - (1) *that they will not give or receive aid in examinations; that they will not give or receive unpermitted aid in class work, in the preparation of reports, or in any other work that is to be used by the instructor as the basis of grading;*
 - (2) *that they will do their share and take an active part in seeing to it that others as well as themselves uphold the spirit and letter of the Honor Code.*
2. *The faculty on its part manifests its confidence in the honor of its students by refraining from proctoring examinations and from taking unusual and unreasonable precautions to prevent the forms of dishonesty mentioned above. The faculty will also avoid, as far as practicable, academic procedures that create temptations to violate the Honor Code.*
3. *While the faculty alone has the right and obligation to set academic requirements, the students and faculty will work together to establish optimal conditions for honorable academic work.*

By signing your name below, you acknowledge that you have abided by the Stanford Honor Code while taking this exam.

Signature: _____

Name: SOLUTIONS

SUNetID: _____

Question	Section 1	2.1	2.2	2.3	3.1	3.2	Total
Score							
Maximum	44	10	15	15	5	11	100

1 Multiple Choice (44 points)

INSTRUCTIONS and TIPS:

- For all of the multiple-choice questions below, **please very clearly mark your answer**. If you have to change your answer, either erase thoroughly (if using pencil) or clearly put an X through your previous answer, and draw an arrow pointing towards the one you intend. **Ambiguous answers will be marked as incorrect.**
- You do not need to justify your answers.
- Some of these multiple-choice problems are **not straightforward**, and you may wish to use scratch paper to work them out. Problems which may require more thought in this section are marked with a *. (Although different things are differently difficult for different people).
- Point values do not necessarily indicate difficulty.

1.1. (5 points) For each of the following quantities, **circle all of the options** that correctly describe the quantity.

(a) (1 pt) The function $f(n)$, where $f(n) = n \log(n)$.

(A) $O(n^2)$ (B) $\Theta(n^2)$ (C) $\Omega(n)$ (D) $O(n)$ (E) $O(\log^2(n))$.

(b) (2 pts) $T(n)$ given by $T(n) = T(n/4) + \Theta(n^2)$ with $T(n) = 1$ for all $n \leq 8$.

(A) $O(n^2)$ (B) $\Theta(n^2)$ (C) $\Omega(n)$ (D) $O(n)$ (E) $O(\log^2(n))$.

By the Master Thm,
this is $\Theta(n^2)$

(c) (2 pts) $T(n)$ which is the running time of the following algorithm:

```
mysteryAlg( n ):
  if n < 3:
    return 1
  return mysteryAlg( n/2 ) + mysteryAlg( (n/2) + 1 )
```

$$T(n) = 2T(n/2) + O(1) \\ = \Theta(n)$$

where above all division is integer division (so a/b means $\lfloor a/b \rfloor$).

(A) $O(n^2)$ (B) $\Theta(n^2)$ (C) $\Omega(n)$ (D) $O(n)$ (E) $O(\log^2(n))$.

1.2. (3 points) Let $G = (V, E)$ be a connected undirected weighted graph, and let T be a minimum spanning tree in G . Decide whether the following statements **must be true** or **may be false**.

(a) (1 pt) For any pair of distinct vertices $s, t \in V$, there is a unique simple path from s to t in T .

True

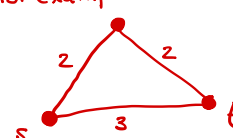
False

(b) (2 pts) For any pair of distinct vertices $s, t \in V$, the cost of a simple path between s and t in T is minimal among all paths from s to t in G .

Simple True
[this was added
before exams
were printed]

False

For example:



1.3. (2 points) **Circle the best way** to finish the sentence:

A top-down dynamic programming algorithm

(A) is exactly the same as a divide-and-conquer algorithm.

(B) is similar to a divide-and-conquer algorithm, except that the top-down dynamic programming algorithm takes advantage of overlapping sub-problems.

(C) is similar to a greedy algorithm, except that a greedy algorithm explores more sub-problems.

(D) is similar to a bottom-up dynamic programming algorithm, except that the bottom-up version stores the results of previously encountered sub-problems while the top-down version does not.

1.4. (3 points) Let $\mathcal{U}$ be a universe of size m , where m is prime, and consider the following two hash families which hash $\mathcal{U}$ into n buckets, where n is much smaller than m . First, consider $\mathcal{H}_1$, which is the set of all functions from $\mathcal{U}$ to $\{1, \dots, n\}$:

$$\mathcal{H}_1 = \{h \mid h : \mathcal{U} \rightarrow \{1, \dots, n\}\}$$

Second, let $p = m$ (so p is prime since we assumed m to be prime), and choose $\mathcal{H}_2$ to be

$$\mathcal{H}_2 = \{h_{a,b} \mid a \in \{1, \dots, p-1\}, b \in \{0, \dots, p-1\}\},$$

where $h_{a,b}(x) = ((ax + b) \bmod p) \bmod n$. You want to implement a hash table using one of these two families. Why would you choose $\mathcal{H}_2$ over $\mathcal{H}_1$ for this task? **Choose the best answer.**

(A) $\mathcal{H}_1$ isn't a universal hash family.

(B) Storing an element of $\mathcal{H}_1$ takes a lot of space.

(C) Storing all of $\mathcal{H}_1$ takes a lot of space.

1.5. (1 point) **Circle True or False.** Let $G = (V, E)$ be a weighted directed graph with a source s and sink t . The cost of a minimum s - t cut in G is equal to the value of a maximum flow from s to t in G .

True

False

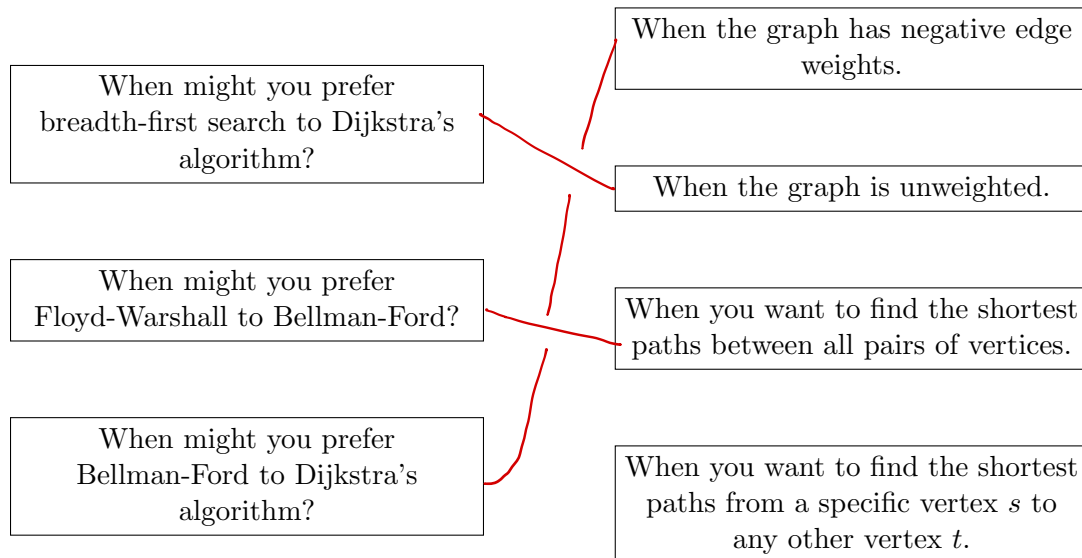
1.6. (1 point) Your friend is trying to prove that a greedy algorithm works by induction, and is struggling with how to formulate the inductive hypothesis. Circle the sentence below that is the **best advice** to give your friend.

(A) Before you come up with the inductive hypothesis, you should first prove the base case.

(B) Try an inductive hypothesis along the lines of, "after you've made j greedy choices, there still exists an optimal solution consistent with the choices you have made so far."

(C) Proof by induction is never a good strategy to prove that a greedy algorithm works.

- 1.7. (3 points) Breadth-first search, Dijkstra's algorithm, the Bellman-Ford algorithm, and the Floyd-Warshall algorithm can all be used to find shortest paths in a graph. **Draw a line from each question to the best answer to that question.**



- 1.8. (4 points) Recall that a *priority queue* is a data structure that holds items with keys. It supports the following operations:

insert(k) inserts an item with key k .

getMin() removes the element with the smallest key and returns it.

For each of the following ways to implement a priority queue, **circle the smallest bound on the worst-case runtime** that you can guarantee for these operations, assuming there are n elements with distinct keys in the priority queue.

- (a) The priority queue is implemented using a **sorted linked list** with pointers to both the beginning and the end.

insert: (A) $O(n)$ (B) $O(\log(n))$ (C) $O(1)$
 getMin: (A) $O(n)$ (B) $O(\log(n))$ (C) $O(1)$

need to scan through to find where k should go

- (b) The priority queue is implemented as a **red-black tree**.

insert: (A) $O(n)$ (B) $O(\log(n))$ (C) $O(1)$
 getMin: (A) $O(n)$ (B) $O(\log(n))$ (C) $O(1)$

- 1.9. *(4 points) Suppose we are given a list of n distinct elements and want to output the largest 10% of them (not necessarily in sorted order). Using a comparison-based algorithm, what is the smallest worst-case runtime we can guarantee to do this? **Circle exactly one answer.**

(A) $O(\log^2 n)$ (B) $O(\log n)$ (C) $O(n)$ (D) $O(n \log n)$

Use SELECT to find the $(0.9) \cdot n^{\text{th}}$ largest element, then scan through + return everything larger.

- 1.10. *(5 points) Consider a citation network with n papers and $m \leq 10 \cdot n$ total citations; that is, the papers are the vertices in an unweighted, directed graph G , and there is a directed edge from i to j if paper i cites paper j . (So G has n vertices and m edges.) For each paper i , in time $O(1)$ you can access the head of a linked list containing all of the indices j so that paper i cites paper j .

Circle all of the problems below which can be solved in time $O(n)$, on **any graph** such that $m \leq 10 \cdot n$.

- (a) Return an array `outDeg` so that `outDeg[i]` is the number of papers that paper i cites.
 (b) Return an array `inDeg` so that `inDeg[i]` is the number of papers that cite paper i .
 (c) Find the strongly connected components in the citation network G .
 (d) Find the length of the shortest directed path from one paper to another paper in the network G .
 (e) Return a list of all pairs (i, j) so that paper i and paper j both cite the same paper k .

DFS twice
to find
SCCs

BFS

$\exists k$ such that

if all the papers cite one paper k^* , then the length of this list might be $\Omega(n^2)$

- 1.11. *(4 points) A minimum spanning tree of a connected weighted undirected graph G is a spanning tree T of G that minimizes the cost $c(T) = \sum_{e \in T} w(e)$, where the sum is over all edges in T and $w(e)$ denotes the weight on an edge e . Define an alternative cost function $\tilde{c}(T) = \max_{e \in T} w(e)$. Decide if the following statements **must be true** for all connected weighted undirected graphs G , or which **may be false** for some connected weighted undirected graphs G .

- (a) Kruskal's algorithm always returns a spanning tree T of G which minimizes $\tilde{c}(T)$.

True

False

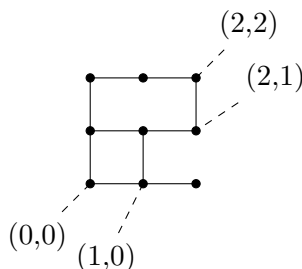
- (b) Prim's algorithm always returns a spanning tree T of G which minimizes $\tilde{c}(T)$.

True

False

Both of these follow exactly as the proofs in class do for MSTs. [Actually, an MST always minimizes $\tilde{c}(T)$, no matter what alg you use to find it.]

- 1.12. *(5 points) Suppose that roads in a city are laid out in an $n \times n$ grid, but some of the roads are obstructed. For example, for $n = 3$, the city may look like this:



where we have only drawn the roads that are not blocked. You want to count the number of ways to get from $(0,0)$ to $(n-1, n-1)$, using paths that only go up and to the right. In the example above, the number of paths is 3.

You decide to use a dynamic programming solution, maintaining an $n \times n$ table M , so that $M[i, j]$ stores the number of ways to get from $(0,0)$ to (i, j) . Suppose that h_{ij} is 1 if there is an unblocked road from $(i-1, j)$ to (i, j) and 0 otherwise; v_{ij} is 1 if there is an unblocked road from $(i, j-1)$ to (i, j) and 0 otherwise. What is the right recurrence relation for this choice of M , for $i, j > 1$? **Circle exactly one.**

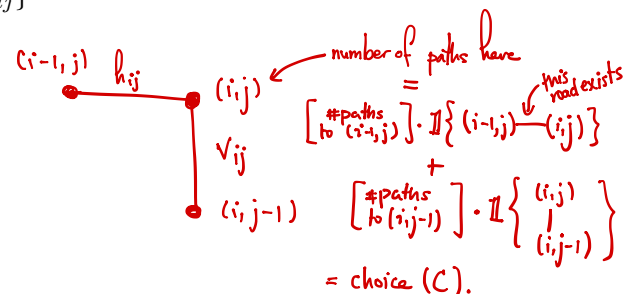
(A) $M[i, j] = \max\{(M[i-1, j] + 1)h_{ij}, (M[i, j-1] + 1)v_{ij}\}$

(B) $M[i, j] = \max\{M[i-1, j] \cdot h_{ij}, M[i, j-1] \cdot v_{ij}\}$

(C) $M[i, j] = \max\{M[i-1, j] + 1, M[i, j-1] + 1\}$

(D) $M[i, j] = M[i-1, j] \cdot h_{ij} + M[i, j-1] \cdot v_{ij}$

(E) $M[i, j] = \max\left\{\sum_{k < i} M[i, k], \sum_{k < j} M[k, j]\right\}$



- 1.13. *(4 points) Suppose that `tryToFindShortestPath(G, s, t)` is a Monte-Carlo randomized algorithm which takes as input a graph G on n vertices, and vertices s and t , and returns a path between s and t .

The path that `tryToFindShortestPath(G, s, t)` returns is a *shortest* path between s and t with probability at least $p(n)$. Here, $p(n)$ is a probability (between 0 and 1) which may depend on n .

Circle exactly one of the options below to fill in the blank in the following sentence with the **smallest answer** that makes the statement correct.

There is a function $A(n) = \dots$ so that if we repeat `tryToFindShortestPath(G, s, t)` $A(n)$ times independently and take the shortest path ever returned, we will find a shortest path between s and t with probability at least 0.99.

(A) $O(1)$

(B) $O(\log(n))$

(C) $O(1/p(n))$

(D) $O(\log(1/p(n)))$

$\mathbb{P}\{\text{don't return a shortest path}\} = (1 - p(n))^{A(n)} \leq e^{-p(n) \cdot A(n)}$, so if $A(n) = \frac{\ln(100)}{p(n)}$, this is ≤ 0.01 .

We can't hope to get away w/ anything smaller, since $\mathbb{E}[\text{\# successes w/ } A(n) \text{ trials}] = A(n) \cdot p(n)$, so if $A(n) = o(1/p(n))$ this is $o(1)$. (little-o means roughly "strictly smaller than")

2 Algorithm Design (40 points)

- 2.1. (10 points) Given an array A of length n , we say that an array B is a *circular shift* of A if there is an integer k between 1 and n (inclusive) so that

$$B = A[k..n] + A[1..k-1],$$

where $+$ denotes concatenation, and where " $A[1..0]$ " denotes the empty array. For example, if $A = [2, 5, 6, 8, 9]$, then

$$B = [6, 8, 9, 2, 5]$$

is a circular shift of A (with $k = 3$). The sorted array A itself is also a circular shift of A (with $k = 1$).

Design a $O(\log(n))$ -time **divide and conquer algorithm** that takes as input an array B which is a circular shift of a sorted array A containing distinct positive integers, and returns the value of the largest element in B . For example, give B as above, your algorithm should return 9.

Write detailed pseudocode for your algorithm below. You do not need to prove that it is correct or justify the running time.

findMax(B):

$n \leftarrow \text{len}(B)$

If $B[1] \leq B[n]$: // case 1
return $B[n]$

$m = \lfloor n/2 \rfloor + 1$

If $B[m] > B[1]$: // case 2
return findMax($B[m..n]$)

If $B[m] < B[1]$: // case 3
return findMax($B[1..m]$)

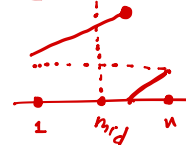
The values are distinct, so we may have this picture in mind:

Case 1:



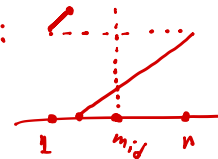
max is at the end.

Case 2:



max is in the 2nd half

Case 3:



max is in the 1st half

(Notice that Case 1 is the base case, since if $n = 1$, then $B[1] \leq B[n]$)

- 2.2. (15 points) There are n final exams today at Stanford; exam i is scheduled to begin at time a_i and end at time b_i . Two exams which overlap cannot be administered in the same classroom; two exams i and j are defined to be *overlapping* if $[a_i, b_i] \cap [a_j, b_j] \neq \emptyset$ (including if $b_i = a_j$, so Exam j starts exactly at the time that Exam i ends). Design a **greedy algorithm** which solves the following problem.

Input: Arrays A and B of length n so that $A[i] = a_i$ and $B[i] = b_i$.

Output: The smallest number of classrooms necessary to schedule all of the exams, and an optimal assignment of exams to classrooms.

For example: Suppose there are three exams, with start and finish times as given below:

i	1	2	3
a_i	12pm	4pm	2pm
b_i	3pm	6pm	5pm

Then the exams can be scheduled in two rooms; Exam 1 and Exam 2 can be scheduled in Room 1 and Exam 3 can be scheduled in Room 2.

Your algorithm should run in time $O(n \log(n) + nk)$, where k is the minimum number of classrooms needed. **Give pseudocode for your algorithm below.** You do not need to prove that it is correct or justify the running time.

scheduleRooms(A, B):
 $n \leftarrow \text{len}(A)$
 $C = [(A[i], i) \text{ for } i=1, \dots, n]$
 $C.sort()$ // increasing by start time $\leftarrow$ also OK to sort by end time
 $rooms = []$
 $endTimes = []$
 for $i=1, \dots, n$:
 for $r=1, \dots, \text{len}(rooms)$:
 if $C[i][1] > endTimes[r]$:
 $rooms[r].append(C[i][2])$
 $endTimes[r] = B[C[i][2]]$
 break
 else: // did not break
 $rooms.append([C[i][2]])$
 $endTimes.append(B[C[i][2]])$
 return $rooms$
 $\uparrow$ list of how exams should be allocated to rooms

- 2.3. (15 points) You are planting tomato plants in a garden, and the garden has n spots arranged in a line. Different spots in the garden will result in different quality tomatoes: suppose that the location i will result in tomatoes of deliciousness $T[i]$, where $T[i]$ is a positive integer. Further, you cannot plant two plants directly next to each other, because they will compete for resources and wilt. Your goal is to create the most deliciousness possible (summed up over all of the tomato plants). **For example**, if the input was $T = [21, 4, 6, 20, 2, 5]$, then you should plant tomatoes in the pattern



and you would obtain deliciousness $21 + 20 + 5 = 46$. You would **not** be allowed to plant tomatoes in the pattern



because there are two tomato plants next to each other.

Design a dynamic programming algorithm which will take as input the array T and return the maximum deliciousness possible given T . Then answer the following questions.

- (a) What are the subproblems that you are using? What is the recursive relationship between subproblems? Briefly (with just a few sentences) explain this recursive relationship.

Let $M[i] =$ max deliciousness obtainable from spots $1, \dots, i$,
assuming we have a plant at i
 $N[i] =$ max deliciousness obtainable from spots $1, \dots, i$,
assuming we don't have a plant at i .

Then

$$M[i] = N[i-1] + T[i]$$

$$N[i] = \max \{ M[i-1], N[i-1] \}$$

This is b/c if we plant at i , then we can't have planted at $i-1$, and so the amount of deliciousness we get is $N[i-1] + T[i]$.

If we didn't plant at i , then we may or may not have planted at $i-1$, so deliciousness is $\max \{ M[i-1], N[i-1] \}$.

[more questions on next page]

Alternative
correct solution:
just keep one array:
 $A[i] = \max \begin{cases} T[i] + A[i-2] \\ A[i-1] \end{cases}$

- (b) Write pseudocode for your algorithm. Your algorithm should take as input the array T , and return a single number which is the maximum amount of deliciousness possible. Your algorithm does not need to output the optimal way to plant the tomatoes.

$\text{maxDelish}(T):$

Initialize $M[1..n]$, $N[1..n]$ to 0's

$M[1] = T[1]$

$N[1] = 0$

for $i = 1, \dots, n:$

$M[i] = N[i-1] + T[i]$

$N[i] = \max \{ M[i-1], N[i-1] \}$

return $\max \{ N[n], M[n] \}$

3 Algorithm Analysis (16 points)

- 3.1. (5 points) Let $C_0, D_0 > 0$ be fixed constants (independent of n). Consider the recurrence relation

$$T(n) = T\left(\left\lfloor \frac{n}{2} \right\rfloor\right) + T\left(\left\lfloor \frac{n}{4} \right\rfloor\right) + C_0 n + D_0,$$

with $T(n) = 1$ for $n = 0, 1, 2, 3, 4$. Rigorously prove, using the substitution method and the definition of $O()$, that

$$T(n) = O(n).$$

We will choose $c = \max\{1, 4(D_0 + C_0)\}$
and show that $T(n) \leq cn \quad \forall n \geq 1$, which, by def of $O(\cdot)$,
will establish $T(n) = O(n)$.

We use the inductive hypothesis that $T(k) \leq c \cdot k \quad \forall 1 \leq k \leq j$.

As a base case, we have

$$T(k) = 1 \leq c \cdot k \quad \forall 1 \leq k \leq 4.$$

This is using the fact that $c \geq 1$.

For the inductive step, assume the inductive hyp. for $j-1$.

Then

$$\begin{aligned} T(j) &= T(\lfloor j/2 \rfloor) + T(\lfloor j/4 \rfloor) + C_0 j + D_0 \\ &\leq \frac{c j}{2} + \frac{c j}{4} + C_0 j + D_0 = j \left(\frac{3c}{4} + C_0 \right) + D_0. \end{aligned}$$

We want this to be $\leq c j$, so we verify

$$\begin{aligned} \Leftrightarrow \left(\frac{3c}{4} + C_0 \right) j + D_0 &\leq c j \\ \Leftrightarrow D_0 &\leq \left(\frac{1}{4} c - C_0 \right) j \end{aligned}$$

$$\Leftrightarrow D_0 \leq \left(\frac{1}{4} c - C_0 \right) \text{ since } j \geq 1$$

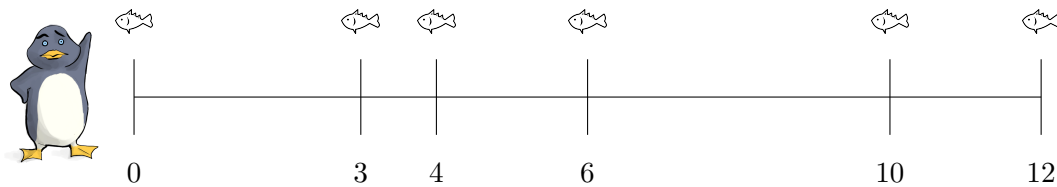
$$\Leftrightarrow 4(D_0 + C_0) \leq c,$$

which is how we chose c above. $\checkmark$

To conclude, the inductive hyp holds for all j . $\checkmark$

- 3.2. (11 points) Plucky the Pedantic Penguin is walking t miles across Antarctica. He needs to eat along the way, but he can only eat when there's a fishing hole for him to catch fish. He can walk at most m miles between meals, and there are n fishing holes along his route.

Plucky is given an array F so that $F[i]$ gives the distance from the start of his journey to the i 'th fishing hole. There are n fishing holes along the way, including at the beginning (so $F[1] = 0$ miles) and the end ($F[n] = t$ miles). For example, the array $F = [0, 3, 4, 6, 10, 12]$, with $t = 12$ corresponds to the setup below:



Plucky wants to stop as few times as possible, given that he can walk at most m miles without eating. (It is okay if he walks exactly m miles between meals). He starts out hungry, so he will always fish at 0 miles; he will also always fish at his destination (at t miles), whether or not he's hungry. In the example above, if $m = 4$, then Plucky should stop 5 times (including his stops at the beginning and the end), for example at 0, 4, 6, 10, 12 miles.

Plucky decides to use the following greedy algorithm:

```

scheduleFishStops( F, m, t ):
    n = length(F); assert F[1] = 0 and F[n] = t
    fishStops = [ F[1] ]
    lastMeal = F[1]
    for i = 2,...,n-1:
        if F[i] - lastMeal <= m and F[i+1] - lastMeal > m:
            fishStops.append( F[i] )
            lastMeal = F[i]
    if t - lastMeal > m:
        return "No way to make it there"
    else:
        fishStops.append(t)
        return fishStops

```

That is, Plucky will hold out for as long as he can, and only stop to fish if he won't be able to make it to the next stop. In the example above, he will stop at 0, 4, 6, 10, 12 miles.

In this problem, you will prove rigorously, by induction, that this strategy is correct.

[Problem 3.2 continued.]

Formally, say that an array S of length r is a **feasible schedule** if S is a sorted array containing r elements of F so that $S[1] = 0$, $S[r] = t$, and for all $i \in \{2, \dots, r\}$, $S[i] - S[i-1] \leq m$. In this problem you will prove the following claim:

Claim. Suppose that F is an array of n strictly increasing positive integers, so that $F[1] = 0$ and $F[n] = t$, and so that for all $i \in \{2, \dots, n\}$, $F[i] - F[i-1] \leq m$. Then `scheduleFishStops(F,m,t)` returns a shortest feasible schedule.

The point of this problem is to demonstrate that you can write a rigorous proof by induction.

- (a) (3 points) State an inductive hypothesis for your proof by induction.

After the j^{th} stop has been added to fishSteps, there exists an optimal feasible schedule S^* so that $S^*[1..j] = \text{fishSteps}$.

- (b) (1 point) Prove the base case.

For $j=1$, $\text{fishSteps}[1] = 0$, and all feasible schedules S^* have $S^*[1] = 0$ by definition.

[Rest of the argument on next page.]

[Problem 3.2 continued.]

(c) (6 points) Prove the inductive step. Make sure you state explicitly what you are proving.

Suppose the inductive hyp holds for $j-1$.

We need to show that it holds for j .

Let S^* be an opt. schedule s.t. $S^*[1..j-1] = \text{fishStops}[1..j-1]$.

Say that we add k as $\text{fishStops}[j]$.

Then $k \geq S^*[j]$, because
$$S^*[j] \leq m + S^*[j-1] = m + \text{fishStops}[j-1],$$

and by construction k is the largest stop so that $k \leq m + \text{fishStops}[j-1]$.

Consider $\tilde{S}^*$ which is obtained from S^* by setting $S^*[j] \leftarrow k$.

This is still feasible, since $\tilde{S}^*[j] = k \leq m + \tilde{S}^*[j-1]$, and

$$\tilde{S}^*[j+1] = S^*[j+1] \leq m + S^*[j] \leq m + k = m + \tilde{S}^*[j].$$

It is still optimal, since $\text{len}(\tilde{S}^*) = \text{len}(S^*)$.

Thus, $\exists$ optimal $\tilde{S}^*$ s.t. $\tilde{S}^*[1..j] = \text{fishStops}[1..j]$, as desired.

(d) (1 point) Prove the conclusion. That is, show that if your inductive argument succeeds, then it implies that **Claim** that you are trying to prove.

Suppose that the inductive hyp holds for $j = r^*$, where r^* is the length of an optimal schedule. Then $\exists S^*$ s.t.

$$\text{fishStops}[1..r^*] = S^*[1..r^*], \text{ aka, } \text{fishStops} = S^*.$$

Thus, the algorithm returns an optimal feasible schedule, as desired..

↑ r^* is the length of S^*

This is the end!

This is blank space for extra work on any problem **to be graded**.
If you use this space, please leave a note on the relevant problem.



Thanks for a great quarter,
and enjoy your summer!!



Technically, summer
doesn't start until
June 20 this year...

Puckey the
Pondantic Penguin

This is blank space for extra work on any problem **to be graded**.
If you use this space, please leave a note on the relevant problem.

This blank page is for **scratch paper**. You may tear it off.
Nothing on this page will be graded.

This blank page is for **scratch paper**. You may tear it off.
Nothing on this page will be graded.